

On the C++¹ Object Programming for Time Series, in the Linux² framework

*George Daniel Mateescu*³

Abstract. We study the implementation of time series through C++ classes, using the fundamentals of C++ programming language, in the *Linux* framework. Such an implementation may be useful in time series modeling.

JEL Classification: C32

AMS Classification: 68N30

Key words: Object Programming, Time Series

Introduction

The C++ programming is widely used in various fields. For time series, or data series, there exists a close structure, namely “*strings*”. Our goal is to explain and implement objects, by using only open source software, such as *Linux – Centos* operating system.

First of all, let us assume that C++ environment is installed and up-to-date:

```
yum clean all
yum install gcc
yum install gcc-c++
```

Second, we have to use a text file editor, such as *vi*.

By using the editor, let us remember the C++ definition of an object class

```
class TS{
public:
    float * s;
    int n;
};
```

¹ C++ language was created by Bjarne Stroustrup

² Linux operating system was created by Linus Benedict Torvalds

³ e-mail: daniel@mateescu.ro

In *Linux – Centos* we will create a file, by using vi editor, vi definition.cpp

```
class TS{
public:
    float * s;
    int n;
};

int main(){
    TS T;
    return 1;
}
```

The above definition is the minimum definition of a C++ class, which stands for a time series. Indeed, the integer *n* is designated to count the number of the values, while the float pointer *s* will describe the values of the string.

Our class will need a constructor, in order to allocate a space for the object. Such a constructor may be:

```
TS::TS(int m){
    n=m;
    s=(float *)malloc(m);
};
```

By using the above constructor, the TS definition of an object will be:

```
TS T(10)
```

which means a time series, T, with 10 values.

For modeling reasons, it may be useful to initialize the values, as for example, a sequence of natural numbers. As consequence, we will declare a second constructor:

```
TS::TS(int m,int p){
    n=m;
    s=(float *)malloc(m);
    for(int i=0;i<m;i++)s[i]=p++;
};
```

Then, the definition:

```
TS I(5,1)
```

means a time series, I, of values: 1, 2, 3, 4, 5.

Class Functions

Some characteristic values may be considered as function members. We will use functions *mean* and *dev* for mean values and standard deviation of a time series, i.e.:

```
class TS{
public:
    float * s;
```

```

int n;
float mean();
float dev();
TS(int);
TS(int,int);
};

```

The member function mean will be:

```

float TS:: mean(){
float m=0;
for(int i=0;i<n;i++)m+=s[i];
return m/n;
};

```

and dev (standard deviation):

```

float TS:: dev(){
float d=0;
for(int i=0;i<n;i++)d+=(s[i]-mean())*(s[i]-mean());
return sqrt(d/n);
};

```

The maximum value and the minimum value may be also implemented as member functions:

```

float TS::min(){
float m=s[0];
for(int i=1;i<n;i++)m>s[i]?m=s[i]:m;
return m;
};

float TS::max(){
float m=s[0];
for(int i=1;i<n;i++)m<s[i]?m=s[i]:m;
return m;
};

```

Operators

The most frequently used operation with time series is the *regression*, i.e. the determination of the linear function

$$\varphi(x) = ax + b$$

which satisfies:

$$\inf_{a,b} \sum_{i=1}^n (y_i - ax_i - b)^2$$

$X(x_i)$ and $Y(y_i)$ being two time series, of n values.

As it is known, parameters a and b are:

$$a = \frac{n \sum_{i=1}^n x_i y_i - \left(\sum_{i=1}^n x_i \right) \left(\sum_{i=1}^n y_i \right)}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2} \quad \text{and} \quad b = \frac{\sum_{i=1}^n x_i^2 \sum_{i=1}^n y_i - \left(\sum_{i=1}^n x_i \right) \left(\sum_{i=1}^n x_i y_i \right)}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2}$$

In our C++ model, we will define the class `regress`, designated to describe the regression parameters.

```
class regress{
public:
    float slope;
    float intercept;
};
```

The regression is implemented by overloading operator `>`

```
regress operator > (TS y, TS x){
    regress r;
    float sx=0, sy=0, sx2=0, sxy=0;
    for(int i=0; i<x.n; i++){
        sx+=x.s[i];
        sy+=y.s[i];
        sx2+=x.s[i]*x.s[i];
        sxy+=x.s[i]*y.s[i];
    };
    r.slope=(x.n*sxy-sx*sy)/(x.n*sx2-sx*sx);
    r.intercept=(sx2*sy-sx*sxy)/(x.n*sx2-sx*sx);
    return r;
};
```

As a consequence, the expression `y>x` will design de regression of y -values depending on x -values. Of course, `x>y`, represent the regression of x -values, depending on y -values.

Extensions

It is possible to extend the operator `>` definition, as for example, by including Durbin Watson test designed to verify the errors autocorrelation.

We remember that Durbin - Watson test is:

$$d = \frac{\sum_{i=2}^n (e_i - e_{i-1})^2}{\sum_{i=1}^n e_i^2}$$

where

$$e_i = y_i - \text{slope} * x_i - \text{intercept}$$

Consequently, we have to extend regress class, by including a new data member:

```
class regress{
public:
    float slope;
    float intercept;
    float DW;
};
```

The new definition of the overloaded operator > will be:

```
regress operator > (TS y, TS x){
    regress r;
    float sx=0, sy=0, sx2=0, sxy=0;
    for(int i=0; i<x.n; i++){
        sx+=x.s[i]; sy+=y.s[i];
        sx2+=x.s[i]*x.s[i]; sxy+=x.s[i]*y.s[i];
    };
    r.slope=(x.n*sxy-sx*sy)/(x.n*sx2-sx*sx);
    r.intercept=(sx2*sy-sx*sxy)/(x.n*sx2-sx*sx);
    float e[x.n];
    for(int j=0; j<x.n; j++)
        e[j]=y.s[j]-r.slope*x.s[j]-r.intercept;
    float e1=0, e2=0;
    for(int k=1; k<x.n; k++) e1+=(e[k]-e[k-1])*(e[k]-e[k-1]);
    for(int p=0; p<x.n; p++) e2+=e[p]*e[p];
    r.DW=e1/e2;
    return r;
};
```

References

Stroustrup, B., 2000. *The C++ Programming Language*. Addison-Wesley Pub Co; 3rd edition (February 15, 2000); ISBN 0-201-70073-5